QsNet^{II}: Defining High-Performance Network Design

QsNetII optimizes interprocessor communication in systems built from standard server building blocks. Its short-message processing unit permits fast injection of small messages, providing ultra-low latency and scalability to thousands of nodes.

Jon Beecroft
David Addison
David Hewson
Moray McLaren
Duncan Roweth
Quadrics

Fabrizio Petrini
Jarek Nieplocha
Pacific Northwest National Laboratory

Cluster computers—parallel computers built from commodity processors—are becoming the predominant supercomputer architecture because of their combined scalable performance and attractive price. As of June 2005, 61 percent of the world's top-500 supercomputers were clusters (http://www.top500.org). This is a significant paradigm shift from a few decades ago, when supercomputers were special purpose, like the Cray vector machines, and designers built them from expensive, custom components.

Clusters that use commodity processors still require high-performance, low-latency networks, if their applications are fine-grained, or if the cluster has many processors. Clusters can use commodity networks, such as Gigabit Ethernet, but these fall short in many scalability and performance aspects.[1]

Consequently, the core of several successful cluster-based supercomputers is a high-performance network. On the one hand, this component interfaces with standard I/O buses, such as peripheral component interconnect (PCI), its extended version (PCI-X), and PCI-Express, thus leveraging commodity computing nodes. On the other hand, it provides scalable performance and cluster aggregation through specialized protocols.[2]

Thus, in a sense, the high-performance network in a cluster computer is the computer because it largely defines achievable performance, widening the range of the applications a cluster can efficiently execute, as well as defining its scalability, fault tolerance, system software, and overall usability.

Because of their key performance-enhancing role, cluster computer networks must meet high standards in four design aspects—performance, scalability, reliability, and programmability. The "Four Critical Design Criteria" sidebar describes these in more detail.

QsNet^{II}, the latest generation Quadrics interconnect, meets these standards, extending previous work on high-performance networks with an aggressive design to achieve ultra-low latency. At the design's core are two ASICs: Elan4 and Elite4. Elan4 is a communication processor that forms the interface between a high-performance multistage network and a processing node with one or more CPUs. Elite4 is a switching component that can switch eight bidirectional communications links, each of which carrying data in both directions simultaneously at 1.3 Gbyte/s. Two virtual channels share the link bandwidth.

Published by the IEEE Computer Society

Four Critical Design Criteria

Cluster computing networks must meet four main criteria.

High performance

A high-performance network is typically (and often, superficially) described in terms of point-to-point latency and bandwidth, although other dimensions—such as the degree of overlap between computation and communication—play an equally important part in cluster supercomputers. Scientific codes often use nonblocking communication, and ideally the network and the network interface can make progress with as little processor overhead as possible. Indeed, several research and industrial projects have shown that the network interface and software communication layer are the major performance bottlenecks.[1]

To reduce this overhead, researchers have proposed several user-level communication models. The designers of Meiko CS-2[2] and Myrinet[3] pioneered many of the techniques that high-performance networks now use, such as zero-copy communication, reliable transmission at the link level, and processing capabilities in the network interface. U-Net tried to provide high-performance, user-level communication in a broad spectrum of networks.[4] Other important research projects in user-level communication are FM,[5] Active Messages,[6] Basic Interface for Parallelism,[7] Virtual Memory-Mapped Communication,[8] and PM.[9] *Computer* recently published a detailed survey of user-level, network interface protocols and related design issues.[10]

Scalability

Many parallel applications display a bulk-synchronous behavior. The processing nodes access the network according to a global, structured communication pattern. They can, for example, execute a global permutation, a gather/scatter phase with a limited set of neighbors, a global synchronization, or a personalized all-to-all information exchange. All these collective patterns are sensitive even to minor bottlenecks—the slowest component will determine the speed of the whole communication pattern—so scalable hardware support for collective communication is a central issue. Recently, hardware support for system software, resource management, parallel file systems, system monitoring, fault-tolerance, and so on has drawn interest among those in the high-performance network community. The common perception is that parallel machines cannot scale to a large number of nodes without hardware support for a few basic mechanisms, such as broadcasting data and performing atomic queries.[11,12]

Reliability

In a machine with thousands of processing nodes, the mean time between failures (MTBF) is typically only a few hours. Thus, the network requires hardware support to detect message corruption through cyclic redundancy checks (CRCs) and to automatically retransmit packets. Although designers could implement these functions at a higher level, for example in the system software, the overhead would be substantial.[13]

Programmability

Message-passing protocols such as the message-passing interface (MPI) and almost all the other system software components require substantial protocol processing when they exchange messages. Offloading much of this overhead to a network processor is an attractive approach to optimizing latency, reducing the protocol's complexity, and effectively overlapping communication with computation. The possibility of programming the network interface at user level, by loading user-level threads without rebooting the machine, provides an extra degree of flexibility that makes it easier to rapidly develop network protocols.

References

1. V. Karamcheti and A.A. Chien, "Do Faster Routers Imply Faster Communication?" *Proc. Parallel Computer and Routing Workshop*, Lecture Notes in Computer Science, vol. 853, Springer-Verlag, 1994, pp. 1-15.
2. M. Homewood and M. McLaren, "Meiko CS-2 Interconnect Elan-Elite Design," *Parallel Computing,* vol. 20, no. 10-11, Nov. 1994, Elsevier, 1994, pp. 1627-1638.
3. N.J. Boden et al., "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, Feb. 1995, pp. 29-36.
4. M. Welsh, A. Basu, and T. von Eicken, "Incorporating Memory Management into User-Level Network Interfaces," 1997, http://www.cs.cornell.edu/tve/u-net/papers/hoti97.pdf.
5. S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," *Proc. Supercomputing Conf.*, vol. 2, IEEE CS Press, 1995; http://portal.acm.org/citation.cfm?id=224360.
6. T. von Eicken et al., "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. Int'l Symp. Computer Architecture*, 1992; http://portal.acm.org/citation.cfm?id=140382.
7. P. Geoffray, L. Prylli, and B. Tourancheau, "BIP-SMP: High Performance Message Passing over a Cluster of Commodity SMPs," *Proc. Supercomputing Conf.*, 1999; http://portal.acm.org/citation.cfm?id=331552.
8. C. Dubnicki et al., "API and Performance of VMMC-2," http://www.cs.berkeley.edu/~culler/hoti97/vmcc.ps.
9. H. Tezuka et al., "PM: An Operating System Coordinated High Performance Communication Library," *Proc. High-Performance Computing and Networking*

continued on p. 4

continued from p. 3

Conf., Lecture Notes in Computer Science 1225, Springer-Verlag, 1997, pp. 708-717.
10. A.F. Raoul, T.R. Bhoedjang, and H.E. Bal, "User-Level Network Interface Protocols," *Computer*, vol. 31, no. 11, Nov. 1998, pp. 53-60.
11. E. Frachtenberg et al., "STORM: Lightning-Fast Resource Management," *Proc. Supercomputing Conf.*, IEEE CS Press, 2002, pp. 1-26.
12. J. Fernández, E. Frachtenberg, and F. Petrini, "BCS MPI: A New Approach in the System Software Design for Large-Scale Parallel Computers," *Proc. Supercomputing Conf.*, IEEE CS Press, 2003, p. 57.
13. R.L. Graham et al., "A Network-Failure-Tolerant Message- Passing System for Terascale Clusters," *Proc. Supercomputing Conf.*, 2002, pp. 77-83.

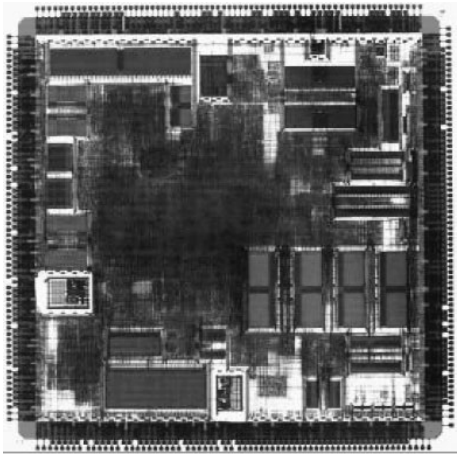

Figure 1. Elan4 is a communication processor that forms the interface between a high-performance multistage network and a processing node. It is 7.5 mm × 7.5 mm, has approximately six million transistors, consumes less than 4W, and is packaged in a 512 enhanced plastic ball grid array (EPBGA).

QsNet^{II} supports broadcast transmission across selected node ranges as well as point-to-point connectivity between arbitrary nodes.

The network's main features are

- *Low latency and high bandwidth*. Elan4 minimizes the latency from the message-passing interface (MPI) by providing specialized units to quickly pipeline small messages into the network, perform protocol processing, and signal completion of the communication. It is designed to use almost all the bandwidth in the PCI-X bus.
- *Scalability to thousands of nodes.* Elan4 is designed to scale to thousands of nodes. Systems like ASC Thunder and ASC Q can execute the most common collective communication primitives (barrier, broadcast, and reduce) in just a few microseconds.[3,4]
- *64-bit internal architecture.* The network can fully support a 64-bit virtual address space.
- *Reliable transmission protocol.* Elan4 implements a reliable transmission protocol in hardware and can thus detect several faults, route the packets around faulty switches, and retransmit packets if they become corrupt.
- *Commodity interface.* The network uses PCI-X, a commodity I/O interface.

Elan4

Figure 1 shows the Elan4. The chip supports 64-bit virtual addresses, generates and accepts packets to and from the network, and provides local processing power to implement the high-level message-passing protocols that parallel processing requires. Figure 2 is a functional diagram of the Elan4 chip. Table 1 gives the features of the major logic blocks.

Elan4's internal architecture lets data flow from the PCI-X bus directly to the output link, thus achieving very low latency and high bandwidth, which is due largely to the STEN processor and two powerful abstractions provided to the communication libraries—the command and input queues.

Protection and latency

A primary design objective for Elan4 was to reduce the latency of protected Unix communication from one processing node to another. Traditionally, the operating system provides that protection as well as multiplexing, both of which add significant latency and overhead to

operating-system-controlled communication. With the Elan4, many different processes can multiplex their operations into the same network hardware at the sending end. Other Elan4s then demultiplex and deliver the data to the correct virtual address space at the receiving end. The chip removes the unnecessary data buffering that almost always adds communication latency. Multicast or broadcast operations can significantly improve the performance of message-passing collective operations, such as scatter, gather, reduce, and barrier synchronization. These types of operations can exploit Elan4's ability to construct complex packets using remote Boolean tests of memory locations, remote input queuing, and remote synchronization. Elan4 has developed a very low-latency method of constructing and issuing these packet types from either a main CPU or from packets that the network receives.

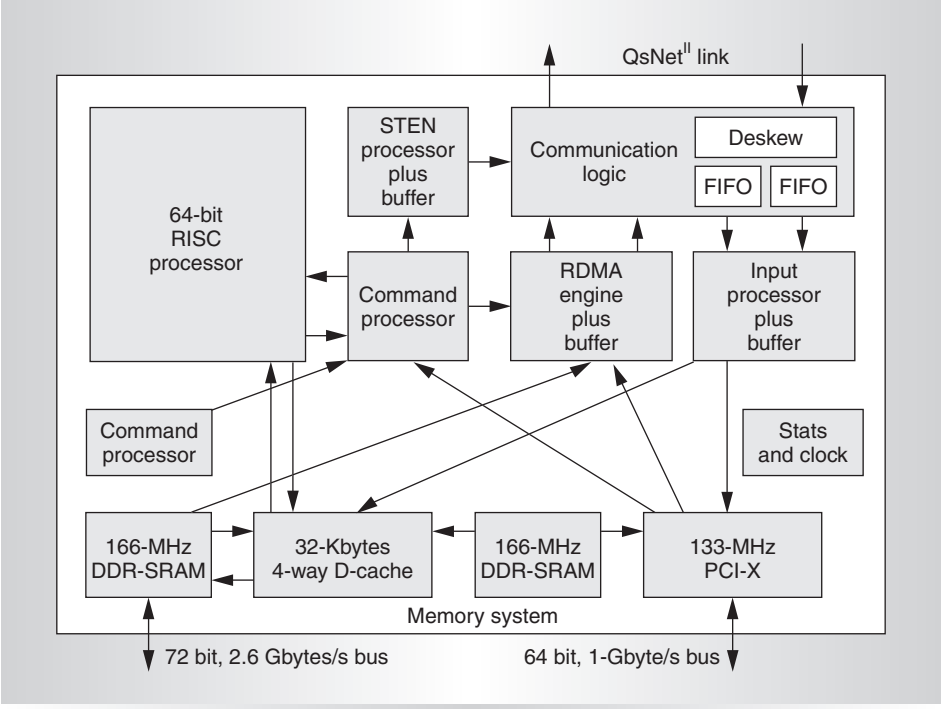

Figure 2. Functional diagram of Elan4. Table 1 lists the functional block characteristics.

Table 1. Primary logic blocks in Elan4 and their characteristics.

Logic block	Characteristics
64-bit RISC microprocessor	Thread processor with 16-Kbyte instruction cache, 64- × 64-bit registers, 65-ns context switching, eight concurrent PCI reads, and one-cycle local load.
Short transaction engine (STEN) processor	Processes short messages, automatic retries, remote Boolean tests, and multicast operations.
Command processor	Defines a virtual command queue interface, uses Q ptr caching and head of Q cache, has rich instruction set, and processes commands at up to 1 Gbyte/s.
Event processor	Process synchronization, writes command Q or main memory.
Input processor	Decodes and executes transactions from network.
32-Kbyte data cache (D cache)	Four-way set-associative; two read ports, one write port, and one cache fill port—all 1.6 Gbyte/s each; up to four pipelined cache fills that speed up accesses to the synchronous, dynamic RAM (SDRAM) interface.
166-MHz DDR SDRAM	72-bit data, including full error-correction code (ECC) logic; 64-bit pipelined interface; 128-byte write buffer; and 128-byte read buffer.
Memory-management unit (MMU)	Translates 64-bit virtual addresses into either 31-bit local SDRAM physical addresses or 50-bit physical addresses. These addresses have four possible values for the top 14 bits of the 64-bit PCI-X master address or a local high-performance command queue address. The MMU has two translation lookaside buffers (TLBs) and 128-page table entries.
Direct memory access (DMA) engine	Two packet assemblers and a pipelined prefetcher.
Communication logic	Network interface connection, 1.3 Gbyte/s each way.
133-MHz PCI-X bus	Read, write bandwidth control; adjacent write block merging; 2-KByte read buffers; up to 12 split reads; big- and little-endian support.

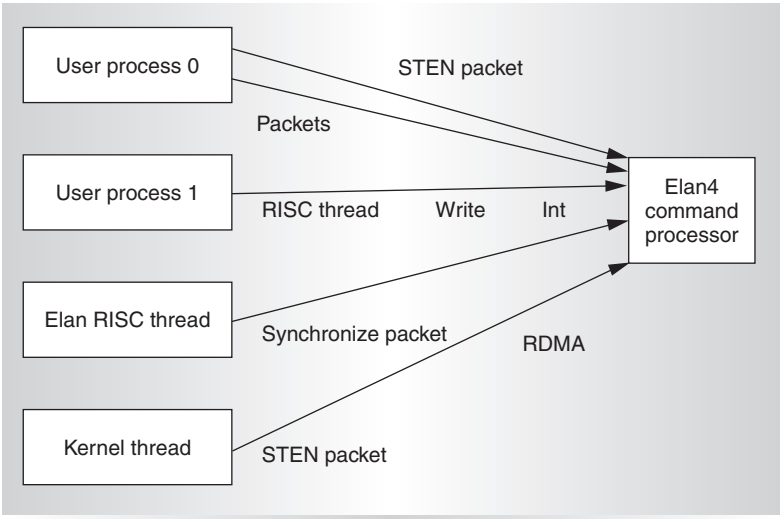

Figure 3. Command-queue model in Elan4. Queue caching allows up to 8,000 simultaneously active command queues. Processes, shown at the left, run at the same time on different CPUs and on the Elan4, and command queues keep data ordered and protected by running in parallel with one another.

Short transaction engine processor

As Figure 2 shows, the STEN processor is closely integrated with the command processor. The main, input, and thread processors use it to assemble short packets for transmission into the network. Quadrics designers optimized STEN for short reads and writes, and for protocol control. It can handle two outstanding packets for each command queue, and packets are pipelined to provide very low latencies.

Packet generation

The PCI-X bus can deliver good bandwidth with long burst operations, but minimizing latency is challenging. There are two ways to take data from a node. The Elan4 can perform master PCI-X read operations on the node's main memory, or the node CPUs can issue writes directly to the PCI-X memory address space. A PCI-X master read typically has a read latency of 750 ns, but it can deliver over 900 Mbytes/s of bandwidth with long burst operations. Direct CPU writes to the PCI-X memory address space have a much lower latency for transferring data values, but the bandwidth can be poor because the writes tend to produce very short bursts.

Some CPU architectures, such as an Intel Itanium Tiger system, can improve this bursting; program stores directed to the PCI-X bus can generate bursts of up to 128 bytes. A bandwidth of 600 Mbytes/s is possible on such an architecture, but only by allowing a relaxed order for the writes, which lets the CPU's write buffer collect several writes. AMD's Opteron can produce 64-byte bursts using relaxed-order mode, which can deliver up to 450 Mbytes/s of PCI-X write bandwidth. However, most CPUs cannot guarantee the write order if a CPU interrupt or an instruction-cache miss occurs.

Elan4 provides two mechanisms for sending data. For large messages, the Elan4 direct memory access (DMA) engine can read from the PCI-X bus at up to 920 Mbytes/s. The DMA engine can pipeline two DMAs, which means that one DMA's completion overlaps the next DMA's startup. This pipelining maintains a high bandwidth between DMAs, but incurs 750 ns of read latency, yielding poor application bandwidth for small messages. This is why Elan4 uses the STEN processor for small messages, as we described earlier.

Command queues

Elan4 has a command processor that manages a set of protected command queues such that any process can write a sequence of commands with data to a command queue. Restricting a queue to a small item size is the lowest-latency method for transferring commands with data from a source to a sink that are operating asynchronously with each other. The source can add data while the queue is not full, the sink can consume data when the queue is not empty, and each can operate at its own rate. With Elan4's queue caching, up to 8,000 command queues can be active at any one time. Because the Elan4 maintains run queues of all active queues and shares the extraction of each queue's commands, there is no practical limit to the command queues that the Elan4 device driver can allocate. Command queues provide protected access to the Elan4 hardware—protection that the operating system call typically provides.

Figure 3 shows the command queue model. A protected process "owns" insertion into the queue, while the Elan4 command processor owns extraction. The command queue appears as a page of PCI-X physical address space, which the Elan4 device driver maps into the user process' virtual address space. All writes into this page from a user process will result in addition-

al data to the corresponding command queue. If the user processes break the command queue's usage rules (about queue overflow, bad addresses, and so on), only that command queue will enter an error state. All other command queues—the system's and those of other user processes—can continue to operate correctly.

This protection is a significant improvement over previous versions of the Elan and other high-performance networks, and it greatly simplifies the device driver because there is no need to quickly deal with an errant process. Elan4's command processor maintains the head of the currently active command queue in a fast on-chip RAM, and the latency through an empty queue is at most two or three cycles. The command processor can recognize out-of-order writes, and it will wait until the gaps are filled before trying to use the data. Consequently, when a CPU receives PCI-X writes, it can use the higher bandwidth relaxed-ordering mode on an Elan4 command queue, yet still process the command queue data in the correct order. Using a page of virtual memory, the Elan4 device driver can map command queues to processes that operate on the Elan4, including the thread and event processors, and network write data. The result is a unified and protected model for starting all Elan4 operations.

Command processor

The command processor initiates operations; it

- generates network packets issued to the STEN processor,
- schedules new DMA descriptors to the DMA processor and new threads to the thread processor,
- executes small copies and small writes,
- controls Elan4 events (synchronization primitives),
- issues interrupts to the main processor, and
- executes simple conditional behavior on the basis of network acknowledgments.

Programmers can combine operations to build more complex ones. If the command queue is generating network packets through the STEN processor, it will automatically handle network retries without further user intervention. The STEN processor will correctly close half-generated packets using a timeout for process scheduling or an interrupt. The command processor will accept simultaneous writes from a node's individual CPUs and local writes from other Elan4 execution units. It will ensure the correct delivery of these writes to their own command queues. It will accept command data at up to 1 Gbyte/s from a node that can achieve long PCI-X programmed I/O (PIO) write bursts. A block of contiguous SDRAM in the Elan4 local memory backs each command queue to be used if Elan4 must automatically resend packets because of network discards. Consequently, there is no need for the queue to rewrite data. Four programmable command queue depths range from 1 to 512 Kbytes.

If a user process running on any of the node's CPUs must write into another node's memory, it can do so simply by writing the command sequence with the data values needed to generate a small write packet directly to a command queue. The procedure requires no operating-system intervention, and the operation is truly protected against other processes—system and user—running on the same CPU or others of the same node. It can take as little as 82 ns from the writes arriving at the Elan4 on the PCI-X bus to the fully formed packet with physical routes and 32-bit CRCs leaving on the Elan4 output link. The packet head can be routing across the network before the tail data has left the main CPU. In this way, the command queues satisfy the very low-latency requirement for small messages.

Event processor

The Elan4's synchronization engine, the event processor, controls the action to be performed when an operation completes, which in turn controls the signaling of an operation's completion. The data structure or "descriptor" held in memory controls the event processor.

When an event fires, the event descriptor defines a copy of data up to 2 Kbytes to be written to a user-defined virtual address. This virtual address could map to a command queue, enabling a very low-latency command sequence to execute in response to network stimulus—without the need to start a thread running on the thread processor. This feature is useful in network scatter/gather operations.

The event descriptors can gather incoming data from the network and then copy it into separate packets routed to several network destinations. Through this method, Elan4 can construct more than four million packets per second and inject them back into the network.

Packet receipt

When Elan4 receives packets from the network, it delivers the data directly into the user's address space without any operating-system intervention, according to a write-based network protocol. All network transactions include a network context number, which Elan4 uses to identify the destination process's virtual address space, and most have a destination 64-bit virtual address value. Elan4 has a full memory management unit (MMU) with two translation look-aside buffers (TLBs) that can hold 128 translations or page table entries (PTEs). It also has a TLB fill engine that will automatically fill the TLBs from large hash tables defined in its local SDRAM. To manage hash conflicts, it uses chained lists from the hash entries. If Elan4 receives a network packet with no valid translation for the hash tables, it will generate an interrupt back to the Elan4 device driver running on a main CPU. The device driver finds the translations, updates the hash table, and restarts the faulted Elan4 process. This page-demanded virtual operation is similar to memory allocation for a normal Unix process and is invisible to the faulted process.

Elan4's MMU translates 14 bits of context and a full 64 bits of address to any 64-bit PCI-X physical address. It supports eight page sizes and lets two sizes be active simultaneously. The Elan4 device driver can define main-memory pages as little or big endian. The TLB fills are pipelined, and other units can continue to use the TLBs during the fills.

Input queues

MPI is a standard communication library common to many parallel applications. With MPI point-to-point communication can take place only with matching sends and receives, in which every send must have a corresponding receive. Problems arise when a sending process posts a send before a receiving process issues the corresponding receive. Elan4 designers had several options for solutions. One was for the receiving process to reject the packets in the hope that the sending process would retransmit them later (after the receiving process issued a receive). The drawback of this approach is that it wastes network bandwidth, and the retry might not occur until long after the receiving process issues the receive, substantially increasing communications latency. Another option was to generate an interrupt to the main processor. The main CPU could then buffer the data and perform a copy to the correct locations when the receiving process issues the receive. This option is also undesirable, however, because an interrupt can be very expensive and disruptive.

A third approach—the one Elan4 uses for MPI—is to buffer and process the posted send data and then perform the final copy after the receiving process issues the receive. This approach adds no load on the network or main processor, and produces the lowest possible latency. Many scientific applications, benefit from such an approach, since they are extremely sensitive to MPI communication latency.[5]

Elan4's input queue construct lets it receive data into a queue in its local memory. When new data goes into the input queue, the input processor (with proper programming) can start a local thread running on Elan4's thread processor. The input queue provides flexible buffering, and the thread processor performs the match operation to connect all sends to their corresponding receives. The thread processor manages the list of posted receives and performs the copy operation required for a posted write.

If a sending process issues a very large send, the MPI code sends only a descriptor to the receiving Elan4. The thread processor can then construct a DMA descriptor, which is posted back to the sending Elan4 so the operation is completed with an efficient DMA write. The thread processor has up to 3.2 Gbytes/s of bandwidth available through block load/store instructions. Any processor can start a thread by writing the program counter and first six registers to a command queue. In as little as 65 ns, the thread scheduler can swap the state of a thread running in one context for the state of another thread running in another context.

The thread processor is closely coupled to the on-chip cache that acts as a data cache (D cache). It uses command queues to inject con-

trol packets into the network.

Elite4 and physical links

Quadrics built the QsNet[II] network using Elite4s arranged in a radix-4 fat-tree network, with each switch having four links down and one to four links up (to higher network stages). Designers selected the fat-tree topology for three reasons: It affords many alternative routes between nodes, the scaling in bisectional bandwidth is linear with network growth, and implementing global network operations, such as broadcast,[6,7] is relatively easy.

Figure 4 shows the Elite4's layout, and Figure 5 shows the optical interface with a 16-way switch. The physical link for Elite4 consists of 10 low-voltage differential signaling (LVDS) pairs in each direction. The communication logic sends a clock with the data and uses it to latch the data on the receiver. It then aligns the data to a local clock derived from the same frequency source but with an arbitrary phase alignment. Each receiver data bit has a digital delay. The delay automatically adjusts during a training process to ensure that the input latch samples the data in the middle of the data eye.

For links longer than 12 m, the network supports a fiber option, which consists of 12-bit parallel optical transceivers and a pair of 12-way parallel fiber ribbons. Because optical receivers require DC-balanced signals, both Elan4 and Elite4 can optionally use 4b5b encoding, which also reduces the range of operating frequency for signal transmission. Table 2 shows peak link performance for unencoded and 4b5b-coded signaling.

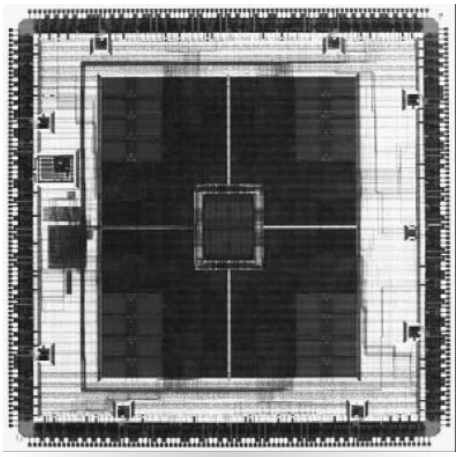

Figure 4. Elite4, an eight-way switch component. We implemented the approximately 8.2 mm × 8.2 mm chip in a 0.18-µm, five-level metal CMOS process; it used a 608-ball EPBGA package and dissipates approximately 6 W.

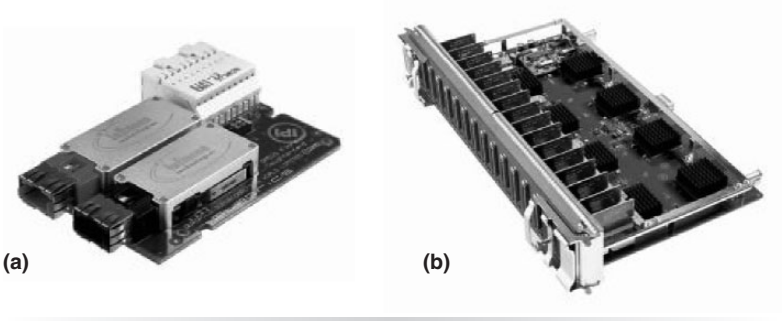

Figure 5. Elite4's optical interface (a) and an interface with a 16-way switch (b).

Table 2. Peak link performance for two types of encoding.

Performance parameter	4b5b	Unencoded
Link data rate (Mbps per wire)	1,333	1,333
Rate after 5/4 (Mbps per wire)	1,066	NA
Peak data (Mbytes/s)	1,066	1,333
Sustainable bandwidth after protocol (Mbytes/s)	928	1,160

Performance evaluation

We have measured QsNet[II]'s performance in a variety of communications and applications benchmarks and parallel applications. In one evaluation, we used a cluster of Intel Itanium 2s with the company's 8870 chipset. In others, we used a variety of platforms.

Basic performance

Figure 6a shows the results of measuring the network's basic bandwidth in a simple ping test, in which two nodes communicate between each other in turn. The network delivered 912 Mbytes/s with the unidirectional ping, representing over 98 percent of the achievable peak bandwidth. It delivered 900 Mbytes/s with bidirectional traffic, which is key to many parallel applications.

In tests on multiple platforms, the MPI short-message latency was only 1.28 µs on the AMD Opteron, and 1.70 µs on the Intel Xeon EM64T, as we describe later. As far as we know, that is the lowest latency any com-

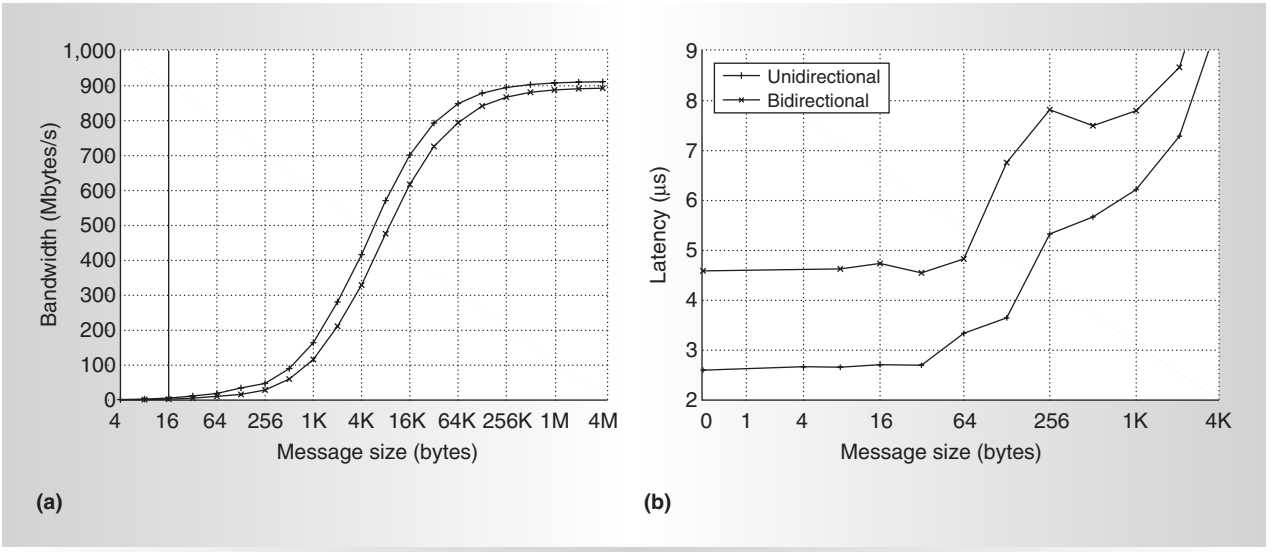

Figure 6. Basic performance in a simple ping test: bandwidth (a) and latency (b).

Table 3. Eight-byte write latency on a 4,096-node machine with 50 meters of cable.

Network or protocol component	Latency (ns)	Total latency (ns)
Cable		200
Delay per meter	4	
50-meter delay	200	
Elite4		231
Switch latency	21	
11 switches (4,096 nodes)	231	
Elan4		240
Input delay	157.5	
Command processor delay	82.5	
MPI tag matching		
Thread processor latency	300	300
Cable + Elite4 + Elan4 + MPI		971

modity, high-performance interconnection network has achieved to date.

Table 3, which gives the write latency breakdown on a 4,096-node machine, shows that the latency is less than one microsecond; the I/O buses take up the rest of the latency.

Figure 7 outlines the steps in the communications protocol. The DMA engine can perform a remote DMA from a source virtual address to a destination virtual address of two processes belonging to the same parallel job, without any intermediate copy or processor intervention. This capability is largely thanks to the MMU in the network interface, which mirrors the one in the host. Quadrics designed the Elan4 DMA engine to run efficiently with the PCI-X interface, and it can issue reads of up to 512 bytes as split transactions to enable pipelining.

The typical execution of a remote DMA has seven steps:

1. The user code, running on a main CPU, generates a DMA descriptor and writes it directly to the PCI-X bus and into a command queue.
2. If the DMA processor is idle, the command processor copies the descriptor into the DMA processor's registers. If it is busy, with another DMA, it copies the descriptor onto a DMA run queue.
3. The DMA prefetch processor takes the virtual source address and issues reads up to 512-byte blocks, using the MMU to translate the virtual addresses into physical main memory addresses. It then wakes up one of the two packet-assembler processors. The prefetch processor continues to fetch DMA data from the PCI-X bus.
4. When the DMA packet assembler wakes up, it fetches the physical-network routes using the virtual destination's process number read from the DMA descriptor. It then starts to generate suitably sized packets for the network. A packet assembler byte-realigns the data streaming

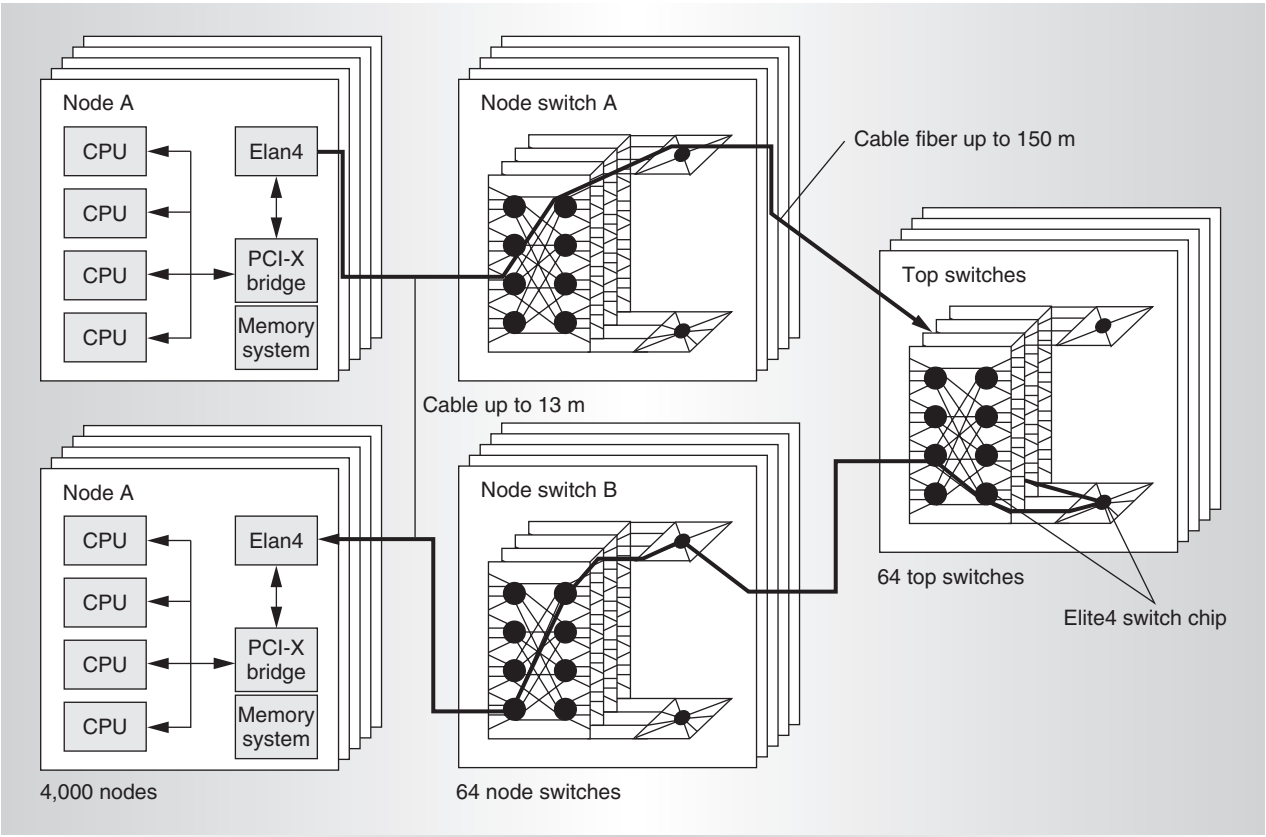

Figure 7. Network communication protocol.

from the source memory system's PCI-X bus and converts it to network transactions, inserting the correct destination virtual address and type.

5. The DMA processor injects the fully assembled packets into the network, where Elite4 chips use and remove route values from the front to direct the packets as they move across the network.
6. When the packets arrive at the destination, the input processor decodes and executes the packet transactions. The MMU converts the virtual destination address to a main-memory physical address. The PCI-X interface merges the separate 128-byte-block transactions into larger block writes to improve the effective write bandwidth.
7. The event processor executes a SetEvent at both the source and destination Elan4 to signal completion of the DMA to other processes. If the event fires, as a result of the SetEvent, the programmer can use this to execute a sequence of commands through a command queue.

As we described earlier, QsNet^{II} provides hardware support for broadcast and barrier operations. Figure 8a shows barrier time as a function of the number of nodes. Additional network stages cause a slight increase on power-of-4 node counts (16, 64, 256, and so on). Figure 8b shows a bisection of the network's bandwidth, which we measured by executing a collective communication pattern, or *complement traffic*. This pattern divides the processing nodes under test into two groups. This is essentially a mirrored ping test, in which each node communicates with its complement located on the other half of the network. The complement traffic forces each message to cross the network bisection.

Both scaling tests show excellent performance. The barrier synchronization takes only 3.5 µs on 512 nodes, while the bisection tests show almost linear scaling. In an actual

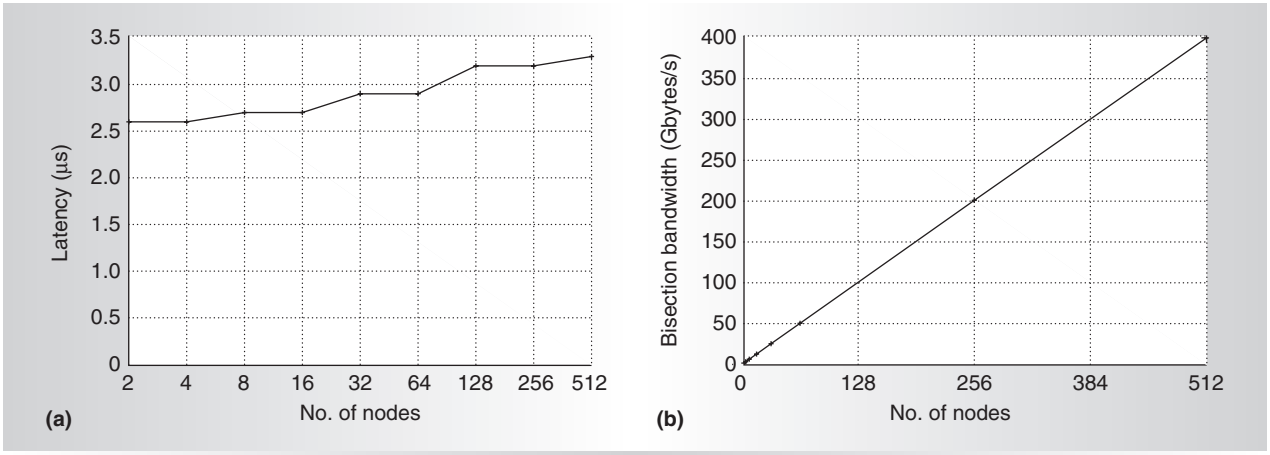

Figure 8. Network scalability: Barrier synchronization (a) and bisection bandwidth (b).

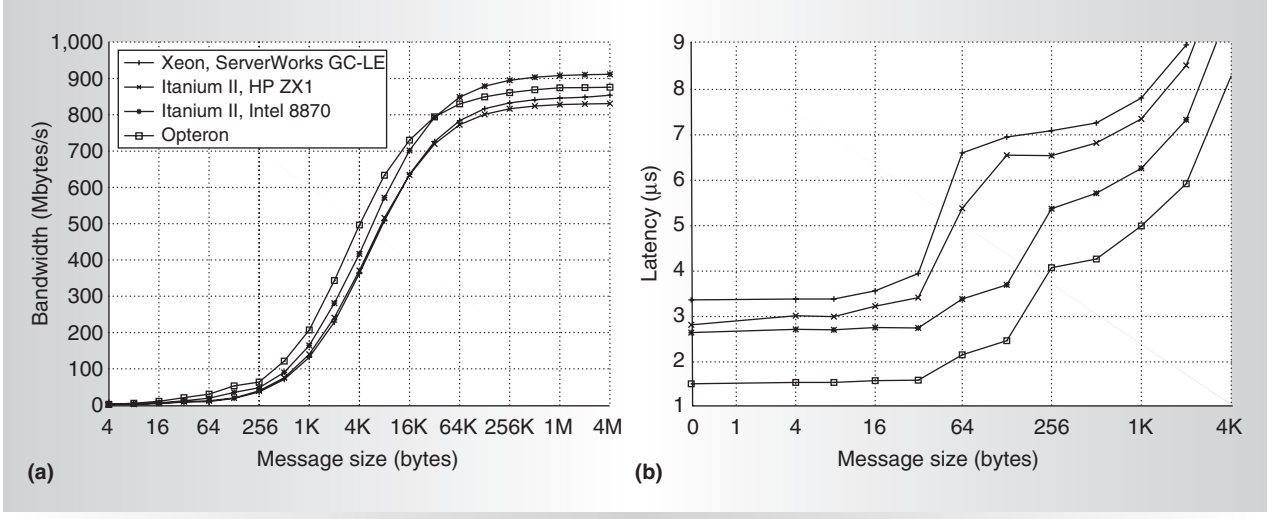

Figure 9. Network's sensitivity to platform: bandwidth (a) and latency (b).

Table 4. Network performance on a variety of platforms.

Platform	Remote DMA (ms)	MPI latency (ms)	MPI bandwidth (Mbytes/s)
HP-DL145 G2-Opteron	0.85	1.28	881.45
Intel SR1400 JR2-Xeon EM64T	1.14	1.70	912.21
Intel SR870BN4-Itanium 2	1.6	2.73	911

system, the PCI-X bridge primarily determines Elan4's peak performance on DMA read operations.

Multiple platforms

Figure 9a shows the results of tests to measure QsNet^{II}'s sensitivity to platform. In these tests, we evaluated the MPI bandwidth for several combinations of processor and motherboard chipsets.

Table 4 lists the performance results QsNet^{II} achieves on platforms with AMD Opteron, Intel Itanium 2, and Intel Xeon EM64T. The highest performance measured is 912 Mbytes/s for the Xeon EM64T host machine.

In any system, many factors influence the actual measure of MPI latency, such as the PIO write bandwidth, the main processor cache's invalidate time, and the bus bridge latency. We evaluated the latency of the three platforms in Table 4 by sending a 0-byte message between two nodes. As the table shows,

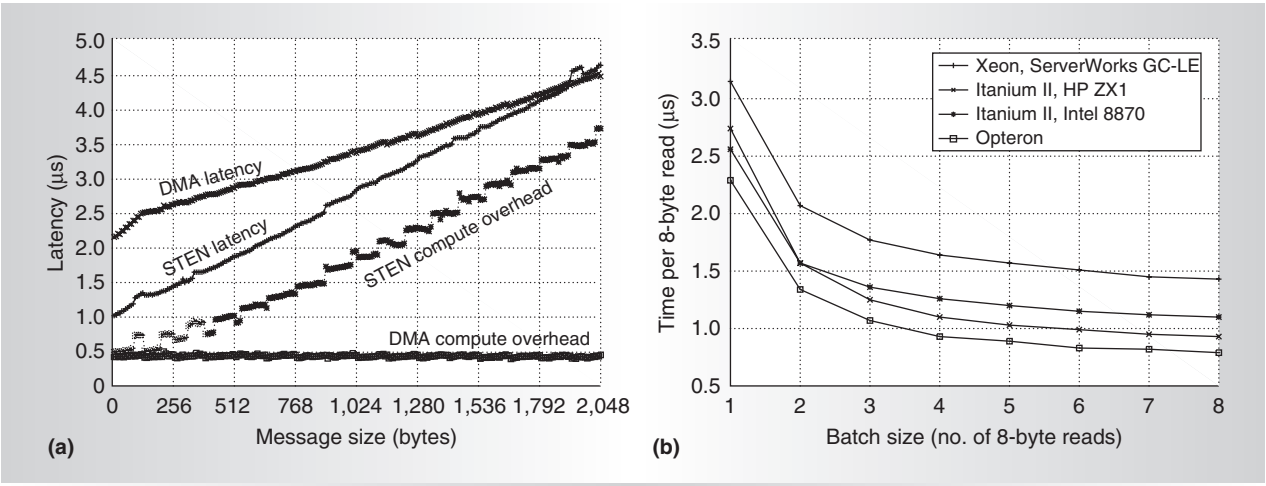

Figure 10. STEN versus DMA performance: put latency (a) and get latency/throughput (b).

the host machine's characteristics play an important role, with latency values ranging from 1.28 to 2.73 µs. AMD's Opteron achieved the lowest latency in large part because of its integrated memory control. For this platform, we tested a dual-CPU system and observed slightly higher latency from the second CPU, which required an additional hop to reach the PCI-X bus bridge.

All the platforms tested were either two- or four-way symmetric multiprocessor (SMP) nodes. Large SMP systems can display greater latencies and greater locality effects. Figure 10a shows how the use of STEN to issue short remote writes reduced short-message latency on the AMD Opteron: an 8-byte remote write latency of 1.0 µs versus 2.2 µs for the equivalent DMA operation. The DMA engine uses the PCI-X bus more efficiently, however, and therefore above 2,000 bytes, DMAs have lower latencies. CPU overhead also figures into the decision of when to switch to DMAs. Issuing a DMA has a fixed overhead of approximately 0.5 µs, but STEN requires the CPU to copy the data, which causes the CPU overhead to be proportional to the message length. STEN also aids in performing a series of short get operations.

As Figure 10b shows, the throughput of single double-word reads to random destinations across the network is 2.3 µs. Batching is one way to increase the rate of issuing double-word reads to discontinuous addresses in the same or different nodes.

Finally Figure 11 shows the performance of a quantum chemistry application, NWChem, running on a variety of platforms. The performance of Elan4-connected Itanium 2 nodes shows a substantial improvement over the same system with Elan3 connectivity, indicating that this code strongly depends on communications performance.

Quadrics designed QsNet^{II} specifically to meet the demands of supercomputing applications. Benchmark results for real-world applications such as NWChem are superior to those for similar processor configurations with different interconnects. Clearly, then, a high-performance interconnect can both shorten application runtime and increase scalability. Processor performance will increase—both through heightened clock rate and as a result of moving to multiple processor cores per device. The challenge for interconnect designers is to track that performance increase in both bandwidth and latency improvements. MICRO

References

1. P. Shivam, P. Wyckoff, and D. Panda, "EMP: Zero-Copy OS-Bypass NIC-Driven Gigabit Ethernet Message Passing," *Proc. Supercomputing Conf.*, IEEE CS Press, 2001, p. 49.
2. J. Liu et al., "Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics," *Proc. Supercomputing Conf.*, IEEE CS Press, 2003, p. 58.
3. F. Petrini et al., "Hardware- and Software-Based Collective Communication on the

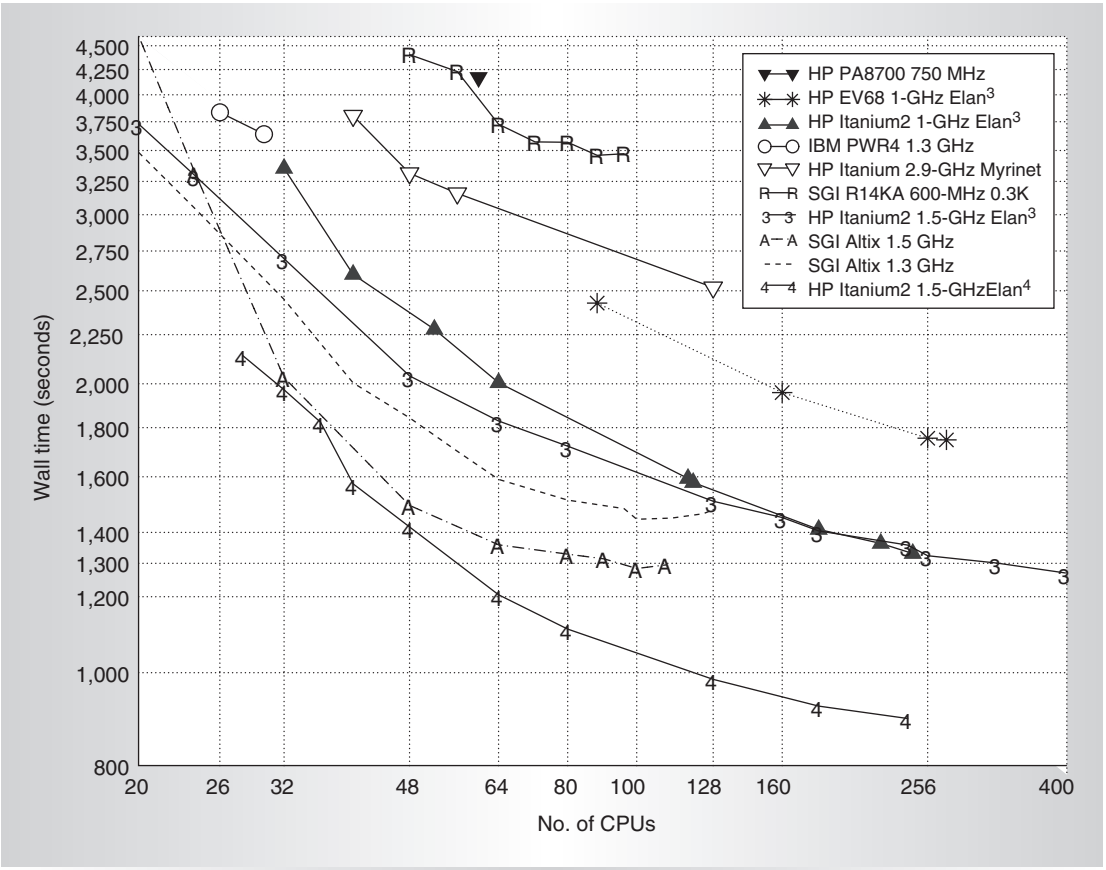

Figure 11. Performance of NWChem on a variety of platforms.

Quadrics Network," *Proc. Int'l Symp. Network Computing and Applications* (NCA 01), IEEE CS Press, 2001, pp. 24-35.

4. D. Roweth, "Optimized Collectives on QsNet^{II}," http://www.quadrics.com/Quadrics/QuadricsHome.nsf/DisplayPages/C51C85B267660ECC80256FBE0014CE8C/$File/collectives.pdf.
5. M. Snir et al., *MPI: The Complete Reference: Volume 1, The MPI Core*, 2nd ed., MIT Press, 1998.
6. M. Homewood and M. McLaren, "Meiko CS-2 Interconnect Elan-Elite Design," *Parallel Computing*, vol. 20, no. 10-11, Nov. 1994, Elsevier, 1994, pp. 1627-1638.
7. F. Petrini et al., "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, Jan.-Feb. 2002, pp. 46-57.

Jon Beecroft is the head of the hardware development team at Quadrics, where he is responsible for the design and development of the ASICs used in QsNet and QsNet^{II}. His research interests include very high-performance general-purpose computing, especially methods for reducing latency to improve scalability. Beecroft has an honors degree in computer engineering from Manchester University.

David Addison is a senior software engineer at Quadrics, where he is responsible for the QsNet^{II} message-passing libraries. His research interests include low-latency message-passing, advanced RDMA protocols and fault-tolerant runtime systems. Addison has a first-class honors degree in computer science from Warwick University.

David Hewson is a senior design engineer at Quadrics, where he is responsible for software architecture for ultra-low latency interconnects and the development of the kernel-messaging system that TruCluster and Lustre use on Quadrics-powered supercomputers. Hewson has a first-class honors degree in mathe-

matics from Durham University. He is a member of the Institute of Mathematics.

Moray McLaren is the R&D manager at Quadrics, where he has overseen the development of the QsNet and QsNet^{II} interconnects. His research interests include high-speed network design and supercomputer architecture. McLaren has a BSc in microelectronics from the University of Edinburgh.

Duncan Roweth is a software research and development manager for Quadrics. His research interests include fault tolerance in parallel systems and the optimization of collectives. Roweth has a PhD in particle physics from Edinburgh University.

Fabrizio Petrini is a laboratory fellow in the Applied Computer Science Group in the Computational Sciences and Mathematics Division at Pacific Northwest National Laboratory. His research interests include various aspects of supercomputers, such as high-performance interconnection networks and network interfaces; job-scheduling algorithms; parallel architectures; operating systems; and parallel-programming languages. Petrini has a Laurea and a PhD in computer science from the University of Pisa, Italy. He has received numerous awards from the Department of Energy for contributions to supercomputing projects, and from other organizations for scientific publications.

Jarek Nieplocha is a laboratory fellow and the technical group leader of the Applied Computer Science Group in the Computational Sciences and Mathematics Division of the Fundamental Science Division at Pacific Northwest National Laboratory. He is also the chief scientist for the High-Performance Computing in Computational Sciences and Mathematics Division. His research interests include optimizing collective and one-sided communication operations on modern networks, runtime systems, parallel I/O, and scalable programming models. Nieplocha received a PhD in electrical and computer engineering from the University of Alabama.

Direct questions and comments about this article to Jon Beecroft, Quadrics Ltd., One Bridewell St., Bristol BS1 2AA, UK; jon@quadrics.com.